Flow Based Min-Cut Balanced Bipartitioning Implementation

Bon Woong Ku
Kyungwook Chang
Kartik Acharya
• Algorithm steps with a small circuit
  – Max-flow computation engine = Dinic’s blocking flow algorithm
• How to handle unreachable nodes
• Our implementation features
  – Speed and accuracy
• Partitioning, and skew impact result
• Possible extensions
1. Parsing and extracting netlist from .blif

- Cell’s weight = 1, Skew = 0%, ratio = 0.5

```plaintext
.model simple
.inputs i0 i1
.outputs o0
.latch i0 a 0
.latch i1 b 0
.names a c 0 1
.names a b d i i 0
.names b e 0 1
.names c d e o 0 i 1
.end
```

- Netlist extraction
  - \( N1(0:2,3), N2(1:3,4), N3(2:5), N4(3:5), N5(4:5) \)
2. Modeling the netlist to the flow network

- \( \rightarrow \) means unit capacity, \( \longrightarrow \) means infinite capacity
- \( \square \) means actual cell nodes, \( \Box \) means added nodes
3. Pick (S,T) pair, and max-flow computation

- Randomly pick a pair of cell node (S,T)
  - S = #3, T = #0
- Find maximum flow network
- Cutsize = 1
• $X = \{1, 2, 3, 4, 5, 6, 8, 9, 10, 11, 12, 13, 14, 15\}$ and $X' = \{0, 7\}$: unbalanced!
• Collapse all nodes in $X'$ to $T$
• $\text{Net}(X,X') = N1(0:2,3)$, adjacent node = #2
5. Max-flow computation

- After collapse #2 to X'
  - $S = #3$, $T = #0,#2,#7$
- Find maximum flow network
- Cutsize = 2
6. Bipartition and collapsing nodes

- $X = \{1,3,4,5,6,8,9,10,12,14,15\}$ and $X' = \{0,2,7,11,13\}$: unbalanced!
- **Collapse all nodes in $X'$ to $T$**
- $\text{Net}(X,X') = N3(2:5)$, adjacent node = #5
7. Max-flow computation

- After collapse #5 to X'
  - \( S = \#3, \ T = \#0, \#2, \#5, \#7, \#10, \#11 \)
- Find maximum flow network
- Cutsize = 3
8. Bipartition and collapsing nodes

• X = \{3, 6, 8, 12\} and X' = \{0, 1, 2, 4, 5, 7, 9, 10, 11, 13, 14, 15\}: unbalanced!
• Collapse all nodes in X to S
• Net(X, X') = N2(1:3, 4) adjacent node = #1, #4
• Choose #1
9. Max-flow computation

- After collapse #1 to X
  - $S = \{#1, #3, #6, #8, #12\}$  $T = \{#0, #2, #5, #7, #10, #11\}$
- Find maximum flow network
- Cutsize = 3
10. Bipartition and collapsing nodes

- \( X = \{1,3,6,8,12\} \) and \( X' = \{0,2,4,5,7,9,10,11,13,14,15\} \): unbalanced!
- Collapse all nodes in \( X \) to \( S \)
- \( \text{Net}(X,X') = N2(1:3,4) \) adjacent node = \#4
11. Max-flow computation

- After collapse #4 to X
  - $S = \#1, \#3, \#4, \#6, \#8, \#12$ $T = \#0, \#2, \#5, \#7, \#10, \#11$
- Find maximum flow network
- Cutsize = 3
12. Final balanced bipartition
Handling unreachable nodes

• There were unreachable node groups in s13207, s9234 circuits
  – We checked this by running DFS and BFS.
  – Result: (G# = group#)
    • s13207: G#1 = 8486, G#2 = 9, G#3 = 71, G#4 = 13, G#5 = 28, G#6 = 107
    • s9234: G#1 = 5787, G#2 = 9, G#3 = 22, G#4 = 13, G#5 = 13

• However, it doesn’t matter in FBB algorithm once we choose the S-T pair in the largest group

1. X is the set of reachable nodes from S through augmenting path.
2. If there are unreachable nodes in the network, it is always considered as X'.
3. Since small partition is collapsed into one node when partition is not balanced, unreachable nodes also would be collapsed into small partition
To make it fast

- We used Dinic’s blocking flow algorithm for max-flow computation
  - It is suitable for sparse flow graph. A cell has limited number of nets
  - Although time complexity is $O(n^2 \log n)$, it’s quite fast in practice

- We handled connectivity, capacity and flow of a graph using STL vectors, Dinic’s algorithm, and collapsing node runs directly on the vectors
  - Using matrix gives us $O(n^3)$
  - Time complexity is now $O(n^2 \log n)$
    - B17 benchmart partitioning runtime comparison
      » Using matrix: 42260s, Our method: 4307s

- We picked up the adjacent node to be collapsed faster
  - We didn’t checked the all possible adjacent node
  - Instead, we checked the first reachable adjacent node
Our implementation feature

• To make it accurate
  – Cycle removal
    • When we collapse the nodes, it could be possible to make invalid flow graph

• With cycle removal, we can reach to node A now

– We completed basic FBB implementation
  • We can give options including skew and balancing ratio to our program
### Best partitioning result

- **Ratio = 0.5, skew = 5%**

<table>
<thead>
<tr>
<th></th>
<th>S13207</th>
<th>S9234</th>
<th>b20_opt</th>
<th>b22_opt</th>
<th>b17_opt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input #</td>
<td>31</td>
<td>36</td>
<td>32</td>
<td>32</td>
<td>37</td>
</tr>
<tr>
<td>Output #</td>
<td>121</td>
<td>39</td>
<td>32</td>
<td>22</td>
<td>46</td>
</tr>
<tr>
<td>Cell #</td>
<td>8696</td>
<td>5808</td>
<td>11979</td>
<td>17351</td>
<td>22854</td>
</tr>
</tbody>
</table>

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Best cut size</td>
<td>98</td>
<td>160</td>
<td>365</td>
<td>980</td>
<td>811</td>
</tr>
<tr>
<td>Partition 1</td>
<td>4185</td>
<td>2759</td>
<td>6208</td>
<td>8683</td>
<td>11539</td>
</tr>
<tr>
<td>Partition 2</td>
<td>4511</td>
<td>3049</td>
<td>5771</td>
<td>8668</td>
<td>11315</td>
</tr>
<tr>
<td>Execution time(s)</td>
<td>4</td>
<td>9</td>
<td>5</td>
<td>100</td>
<td>140</td>
</tr>
</tbody>
</table>

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Worst cut size</td>
<td>375</td>
<td>530</td>
<td>2260</td>
<td>3662</td>
<td>3678</td>
</tr>
<tr>
<td>Partition 1</td>
<td>4131</td>
<td>3020</td>
<td>5720</td>
<td>8244</td>
<td>11452</td>
</tr>
<tr>
<td>Partition 2</td>
<td>4565</td>
<td>2788</td>
<td>6259</td>
<td>9107</td>
<td>11402</td>
</tr>
<tr>
<td>Execution time(s)</td>
<td>98</td>
<td>19</td>
<td>122</td>
<td>327</td>
<td>474</td>
</tr>
</tbody>
</table>

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Average cut size</td>
<td>204</td>
<td>271</td>
<td>1390</td>
<td>2047</td>
<td>1726</td>
</tr>
<tr>
<td>Execution time(s)</td>
<td>19</td>
<td>11</td>
<td>91</td>
<td>192</td>
<td>228</td>
</tr>
<tr>
<td>Cell# / (cut size X time)</td>
<td>2.24</td>
<td>1.95</td>
<td>0.09</td>
<td>0.04</td>
<td>0.06</td>
</tr>
</tbody>
</table>
Skew impact result

- **Benchmark:** S9234, #cell = 5808

**CUT SIZE RESULT**

- Best cutsize: 172, 478, 520, 414, 333
- Worst cutsize: 110, 359, 297, 245, 195
- Average cutsize: 116, 224, 218, 147, 156

**AVERAGE RUNTIME RESULT**

- Average runtime: 0%, 5%, 10%, 20%

**Legend:**
- Best cutsize
- Worst cutsize
- Average cutsize
- Linear (average cutsize)
Expecting extensions

- Compare with KL, FM partitioning result
- Show the impact of s-t selection on cutsize
  - Random s-t selection
  - Max-min-path s-t selection using modified Dijkstra’s algorithm
    - Modified Dijkstra’s algorithm finds min-path to all other nodes from source, with same time complexity of Dijkstra’s algorithm
    - We choose a sink which has maximum min-path from source